MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Memo No. 427 June 1977

EXPLICIT CONTROL OF REASONING

by

Johan de Kleer, Jon Doyle*,
Guy L. Steele Jr.**, and Gerald Jay Sussman

Abstract:

The construction of expert problem-solving systems requires the development of techniques for using modular representations of knowledge without encountering combinatorial explosions in the solution effort. This report describes an approach to dealing with this problem based on making some knowledge which is usually implicitly part of an expert problem solver explicit, thus allowing this knowledge about control to be manipulated and reasoned about. The basic components of this approach involve using explicit representations of the control structure of the problem solver, and linking this and other knowledge manipulated by the expert by means of explicit data dependencies.

* Fannie and John Hertz Foundation Fellow
** NSF Fellow

This research was conducted at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-75-C-0643.

Acknowledgements:

We thank Charles Rich, Marvin Minsky, Drew McDermott, Richard Stallman, Marilyn McLennan, Richard Brown, Peter Szolovits, and Gerald Roylance for suggestions, ideas and comments used in this paper. Jon Doyle is supported by a Fannie and John Hertz Foundation graduate fellowship. Guy Steele is supported by a National Science Foundation graduate fellowship. This research was conducted at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-75-C-0643.

This paper will be presented at the ACM Conference on Artificial Intelligence and Programming Languages, Rochester, New York, August 1977.

Contents:

The Problem

A goal of Artificial Intelligence is to construct an "advice taker",[Advice Taker] a program which can be told new knowledge and advised about how that knowledge may be useful. One approach toward achieving this goal has been to use additive formalisms for the representation of knowledge. Those formalisms derived from mathematical logic have been the most popular. Unfortunately, the resulting systems are combinatorially explosive. It is difficult to provide incremental guidance for problem solvers which use these additive formalisms because of the unsolved question of how to describe knowledge about how to profitably use other knowledge.[Additive Knowledge]

Substantial progress has been made in constructing expert problem solvers for limited domains by abandoning the goal of incremental addition of knowledge. Experts have usually been constructed as procedures whose control structure embodies both the knowledge of the problem domain and how it is to be used. The "procedural embedding of knowledge"[Procedures] paradigm seems natural for capturing the knowledge of experts because of the apparent coherence we observe in the behavior of a human expert who is trying to solve a problem. For each specific problem he seems to be following a definite procedure with discrete steps and conditionals. In fact, an expert will often report that his behavior is controlled by a precompiled procedure. One difficulty with this theory is the flexibility of the expert's knowledge. If one poses a new problem, differing only slightly from one which we have previously observed an expert solve, he will explain his new solution as the result of executing a procedure differing in detail from the previous one. It really seems that the procedure is created on the fly from a more flexible base of knowledge.

We believe that the procedural explanation is an artifact of the explanation generator rather than a clue to the structure of the problem solving mechanism. The apparently coherent behavior of the problem solver may be a consequence of the individual behaviors of a set of relatively independent agents.[Coherence] As an example of coherent behavior on the part of a problem solver constructed from incoherent knowledge sources, we cite the operation of the EL electronics circuit analysis program.[EL] EL is constructed from a set of independent demons, each implementing some facet of electrical laws applied to particular device types. The nature of knowledge in the electrical domain is such that the analysis of a particular circuit is highly constrained, and so the traces of performance and explanations produced by EL are coherent. Like Simon's ant,[Simon's Ant] EL displays complex and directed behaviors which are largely determined by the nature of the terrain, that is, the circuit.

Our Approach

We here present a problem solving methodology by which the individual behaviors of a set of independent rules are coordinated so as to exhibit coherent behavior.

This methodology establishes a set of conventions for writing rules, and a set of features which the rule interpreter must supply to support these conventions. Our rules operate on a data base of facts. Each fact is constructed with a justification which describes how that fact was deduced from antecedent facts.

The key to obtaining coherence is explicitly representing some of the knowledge which is usually implicit in an expert system:

> We explicitly represent the control state of the problem solver. For example, each goal is asserted and justified in terms of other goals and facts. We distinguish various kinds of goals for deduction and action to which different subsets of rules apply. These explicit goals and their justifications are used in reasoning about the problem solver's actions and its reasons for decisions.[Explicit Control]
>
> We explicitly represent as facts knowledge about how other facts are to be used. In traditional methods of representing knowledge the way a piece of knowledge is used is implicit rather than something that can be reasoned about. In PLANNER, for example, the use of a piece of knowledge is fixed at the time that the knowledge is built into the problem solver, and it is not possible to later qualify the use of this knowledge. One can specify a rule to be used as either a consequent or antecedent theorem, but one can not later say "But don't do that if ... is true." To allow the expression of such statements some facts must be assertions about other facts.
>
> We explicitly represent the reasons for belief in facts. Each fact has associated justifications which describe reasons for believing that fact and how they depend on beliefs in other facts and rules. A fact is believed if it has well-founded support in terms of other facts and rules.[Well-Founded Support] The currently active data base context is defined by the set of primitive premises and assumptions in force.
>
> The justifications can be used by both the user and the problem solver to gain insight into the operation of the set of rules on a particular problem. One can perturb the premises and examine the changed beliefs that result. This is precisely what is needed for reasoning about hypothetical situations. One can extract information from the justifications in the analysis of error conditions resulting from incorrect assumptions. This information can be used in dependency-directed backtracking[Backtracking] to pinpoint the faulty assumptions and to limit future search.
>
> The explicit data dependencies allow us to control the connection between control decisions and the knowledge they are based on.[Separation] We can separate the reasons for belief in derived facts from the control decisions affecting their derivation when the facts are independent of the control decisions. Anomalous dependencies are produced when this separation is not made. In chronological backtracking[Micro-PLANNER] control decisions are confused with the logical grounds for belief in facts. This results in the loss of useful

information when control decisions are changed.

This technique of using explicit control knowledge to guide a problem solver does not resolve all difficulties, since it is often unclear as to what knowledge is usefully made explicit. In the following we present examples of the use of explicit control knowledge in constructing coherent behaviors from incoherent knowledge sources.Control Vocabulary

Explicit Control Assertions

Suppose we know a few simple facts, which we can express in a bastard form of predicate calculus:

```
(-> (human :x) (fallible :x))    Every human is fallible!
(human Turing)                   Poor Turing.
```

If provided with a simple syntactic system with two derivation rules (which we may interpret to be the conjunction introduction and modus ponens rules of logic),

```
A                 (-> A B)
B                 A
---------         --------
(AND A B)         B
```

then by application of these rules to the given facts we may derive the conclusion

```
(AND (fallible Turing) (human Turing)).
```

Since the rules are sound, we may believe this conclusion.

Several methods can be used to mechanically derive this conclusion. One scheme (the British Museum Algorithm) is to make all possible derivations from the given facts with the given rules of inference. These can be enumerated breadth-first. If the desired conclusion is derivable, it will eventually appear and we can turn off our machine.

The difficulty with this approach is the large number of deductions made which are either irrelevant to the desired conclusion (they do not appear in its derivation) or useless, producing an incoherent performance. For instance, in addition to the above, conjunction introduction will produce such wonders as:Suppression

```
(AND (human Turing) (human Turing))
(AND (-> (human :y) (fallible :y)) (human Turing))
```

The literature of mechanical theorem-proving has concentrated on sophisticated

deductive algorithms and powerful but general inference rules which limit the combinatorial explosion. These combinatorial strategies are not sufficient to limit the process enough to prevent computational catastrophe. Verily, as much knowledge is needed to effectively use a fact as there is in the fact.

Consider the problem of controlling what deductions to make in the previous example so that only relevant conjuncts are derived. The derivation rules can be modified to include in the antecedent a statement that the consequent is needed:

```
(SHOW (AND A B))          (SHOW B)
A                         (-> A B)
B                         A
--------------            -------
(AND A B)                 B
```

Given these rules, only relevant conclusions are generated. The assertion (SHOW X) says nothing about the truth or falsity of X, but rather indicates that X is a fact which should be derived if possible. Since the "SHOW" rules only deduce new facts when interest in them has been asserted, explicit derivation rules are needed to ensure that if interest in some fact is asserted, interest is also asserted in appropriate antecedents of it. This is how subgoals are generated.[Implications]

```
(SHOW (AND A B))          (SHOW (AND A B))
----------------          A
(SHOW A)                  ----------------
                          (SHOW B)
```

```
(SHOW B)
(-> A B)
-------
(SHOW A)
```

With these rules the derivation process is constrained. To derive

(AND (fallible Turing) (human Turing)),

interest must be first asserted:

(SHOW (AND (fallible Turing) (human Turing))).

Application of the derivation rules now results in the following sequence of facts:

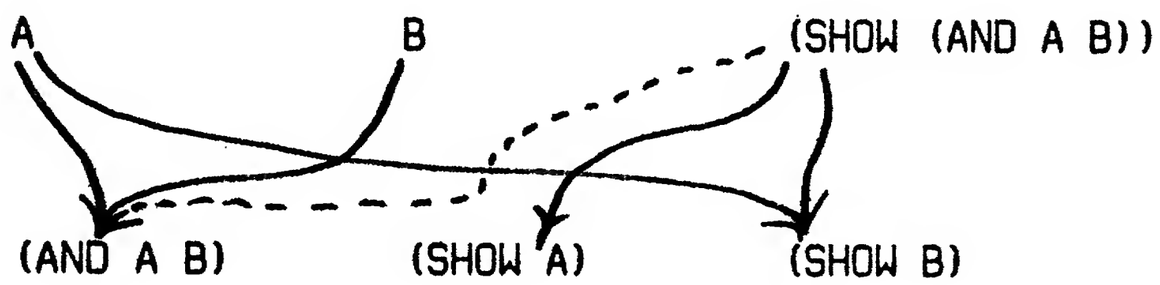
```
(SHOW (fallible Turing))
(SHOW (human Turing))
(fallible Turing)
(AND (fallible Turing) (human Turing))
```

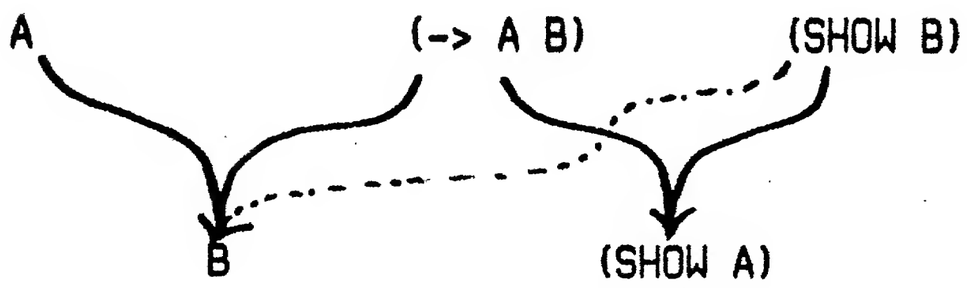
These are absolutely all the facts that can be derived, and no facts were derived which were not relevant to the goal.

Explicit Data Dependencies

This apparent coherence has been achieved by the manipulation of explicit control assertions. The use of explicit control necessitates the use of explicit dependencies. If the conclusions of a rule of inference uniformly depended on the antecedents of the rule then SHOW rules would cause belief in their consequents to depend on the statement of interest in them. That is wrong. If the truth of a statement depends on the truth of the need for it, the statement loses support if interest in it is withdrawn. Even worse, if a derived conclusion is inconsistent, one might accidently blame the deducer for his curiosity instead of the faulty antecedent of the contradiction! The dependence of each new conclusion on other beliefs must be made explicit so that the dependencies of control assertions can be separated from the reasons for derived results. In the conjunction introduction rule, the truth of the conjunction depends only on the truth of the conjuncts but interest in the truth of the conjuncts propagates from interest in the truth of the conjunction.

We can depict the rules of inference for truth and control with the correct dependency information as follows:





In this diagram, the target statement is derived if the source statements are known. Only the solid arrows represent dependency links.

The AMORD language

Primitives

To provide notation for expressing the explicit control and dependency structure of the problem solving process, we have developed an antecedent reasoning system called AMORD.[AMORD] AMORD is a language for expressing pattern-invoked procedures, which monitor a pattern-indexed data base, coupled with a system for automatic maintenance of dependency information. The basic AMORD constructs are RULEs and ASSERTions.

New facts can be inserted into the data base with

(ASSERT <pattern> <justification>),

where any variables in the arguments inherit their values from the lexically surrounding text, and <justification> is a specification of the reason for belief in the fact specified by <pattern>. The justification is constructed from {1} an arbitrary (possibly composite) name denoting the justification type (often the name of a rule), and {2} the factnames of the assertions on which the belief depends. Variables are denoted by atoms with a ":" prefix. Each fact[Variants] has a unique factname.

A rule is a pattern-invoked procedure, whose syntax is:

(RULE (<factname> <pattern>) <body>),

where <factname> is a variable which will be bound to the factname of any fact which unifies[Unifies] with <pattern>, and <body> is a set of AMORD forms to be evaluated in the environment specified by adding the variable bindings derived from the unification and the binding of <factname> to those derived from the lexical environment of the rule. The primary use of <factname> is in specifying justifications for ASSERTs in the body. Rules are run on all matching facts.[Order]

Sometimes it is necessary to assume a truth "for the sake of argument". Such a hypothetical fact is used when we wish to investigate its consequences. Perhaps it is independently justifiable, but it is also possible that it is inconsistent with other beliefs and will be ruled out by a contradiction. We construct such a hypothetical assertion using ASSUME:

(ASSUME <pattern> <justification>)

Here <justification> provides support for the <u>need</u> for the assumption, not the assumed

fact. If the assumed fact is contradicted and removed by backtracking, the negation of the assumed fact is asserted and supported by the reasons underlying the contradiction.[Nogood]

Examples

The forward version of conjunction introduction is implemented in AMORD as the following rule:

```
(Rule (:f :a)
      (Rule (:g :b)
            (Assert (AND :a :b) (&+ :f :g))))
```

To paraphrase this rule, the addition of a fact f with pattern a into the data-base results in the addition of a rule which checks every fact g in the data-base and asserts the conjunction of a and the pattern b of g. Thus if A is asserted, so will be (AND A A), (AND A (AND A A)), (AND (AND A A) A), etc. Note that the atom AND is not a distinguished symbol.

To control these deductions, the above rule can be replaced by the following rule which effects consequent reasoning about conjunctive goals.

```
(Rule (:g (SHOW (AND :p :q)))
      (Rule (:c1 :p)
            (Rule (:c2 :q)
                  (Assert (AND :p :q) (&+ :c1 :c2)))
            (Assert (SHOW :q) ((BC &+) :g :c1)))
      (Assert (SHOW :p) ((BC &+) :g)))
```

In this rule the control statements (SHOWs) depend on belief in the relevant controlled facts so that the existence of a subgoal for the second conjunct of a conjunctive goal depends on the solution for the first conjunct. At the same time, no controlled facts depend on control facts, since the justification for a conjunction is entirely in terms of the conjuncts, and not on the need for deriving the conjunction. This means that the control over the derivation of facts cannot affect the truth of the derived facts. Moreover, the hierarchy of nested, lexically scoped rules allows the specification of sequencing and restriction information. For instance, the above rule could have been written as

```
(Rule (:g (SHOW (AND :p :q)))
      (Rule (:c1 :p)
            (Rule (:c2 :q)
                  (Assert (AND :p :q) (&+ :c1 :c2))))
      (Assert (SHOW :p) ((BC &+) :g))
      (Assert (SHOW :q) ((BC &+) :g)))
```

This form of the rule would also only derive correct statements, but would not be as tightly controlled as the previous rule. In this case, both subgoals are asserted immediately, although there is no reason to work on the second conjunct unless the first conjunct has been solved. This form of the rule allows more work to be done in that the possible mutual constraints of the conjuncts on each other due to shared variables is not accounted for. That is, in the first form of the rule, solutions to the first conjunct were used to specialize the subgoals for the second conjunct, so that the constraints of the solutions to the first are accounted for in the second subgoal. In the second form of the rule much work might be done on solving each subgoal independently, with the derivation of the conjunction performed by an explicit matching of these derived results. This allows solutions to the second subgoal to be derived which cannot match any solution to the first subgoal.

Other consequent rules for Modus Ponens, Negated Conjunction Introduction, and Double Negation Introduction are similar in spirit to the rule for Conjunction Introduction:

```
(Rule (:g (SHOW :q))
      (Rule (:i (-> :p :q))
            (Rule (:f :p)
                  (Assert :q (MP :i :f)))
            (Assert (SHOW :p) ((BC MP) :g :i))))

(Rule (:g (SHOW (NOT (AND :p :q))))
      (Rule (:t (NOT :p))
            (Assert (NOT (AND :p :q)) (-&+ :t)))
      (Rule (:t (NOT :q))
            (Assert (NOT (AND :p :q)) (-&+ :t)))
      (Assert (SHOW (NOT :p)) ((BC -&+) :g))
      (Assert (SHOW (NOT :q)) ((BC -&+) :g)))

(Rule (:g (SHOW (NOT (NOT :p))))
      (Rule (:f :p)
            (Assert (NOT (NOT :p)) (--+ :f)))
      (Assert (SHOW :p) ((BC --+) :g)))
```

The BLOCKS World

We will discuss problem solving in the blocks world as an example of our methodology. First, we formalize the domain with a set of logical axioms which express a McCarthy-Hayes situational calculus.[Situations] We use the syntax (TRUE <statement> <situation>) to state that the indicated statement holds in the indicated situation. TRUE is

a syntactic convenience. We could just as well add an extra argument to each of the predicates of the domain. For example, the following axiom expresses the fact that in the situation arrived at after a PUTON operation the block which moved is on the block it was put on.

```
(Assert (-> (AND (TRUE (CLEARTOP :x) :s)
                 (TRUE (SPACE-FOR :x :y) :s))
            (TRUE (ON :x :y) ((PUTON :x :y) . :s)))
        (Premise))
```

More axioms are needed for the blocks world. Blocks not moved by a PUTON remain on their former support:

```
(Assert (-> (AND (TRUE (ON :a :b) :s) (NOT (= :a :x)))
            (TRUE (ON :a :b) ((PUTON :x :y) . :s)))
        (Premise)).
```

A block is said to be CLEARTOP if no other block is ON it. We assume for simplicity that only one block can be ON another, and introduce statements of CLEARTOP for blocks made clear by PUTON:

```
(Assert (-> (AND (TRUE (ON :x :b) :s)
                 (AND (NOT (= :b Table))  (NOT (= :y :b))))
            (TRUE (CLEARTOP :b) ((PUTON :x :y) . :s)))
        (Premise)).
```

If a block is CLEARTOP, it remains so after any action which does not place another block ON it.

```
(Assert (-> (AND (TRUE (CLEARTOP :b) :s) (NOT (= :y :b)))
            (TRUE (CLEARTOP :b) ((PUTON :x :y) . :s)))
        (Premise))
```

A block can be ON only one other block.

```
(Assert (-> (AND (TRUE (ON :x :z) :s) (NOT (= :z :y)))
            (NOT (TRUE (ON :x :y) :s)))
        (Premise))
```

The definition of CLEARTOP is:

```
(Assert (-> (TRUE (ON :x :y) :s)
            (NOT (TRUE (CLEARTOP :y) :s)))
        (Premise)).
```

If a block is CLEARTOP, it has SPACE-FOR any other block.

```
(Assert (-> (TRUE (CLEARTOP :x) :s)
            (TRUE (SPACE-FOR :y :x) :s))
        (Premise))
```

If a block is not CLEARTOP, it does not have SPACE-FOR anything more. This assumes only one block can be ON another.

```
(Assert (-> (AND (NOT (TRUE (CLEARTOP :x) :s)) (NOT (= :x Table)))
            (NOT (TRUE (SPACE-FOR :y :x) :s)))
        (Premise))
```

The table always has SPACE-FOR everything.

```
(Assert (TRUE (SPACE-FOR :x Table) :s) (Premise))
```

We set up an initial state of the system by adding situation-specific axioms.

```
(Assert (TRUE (ON C A) INIT) (Premise))
(Assert (TRUE (ON A Table) INIT) (Premise))
(Assert (TRUE (ON B Table) INIT) (Premise))
(Assert (TRUE (CLEARTOP C) INIT) (Premise))
(Assert (TRUE (CLEARTOP B) INIT) (Premise))
```

Problem Solver Strategies

There are a number of strategies for using this description of this blocks world for problem solving. Consider the problem of finding a sequence of actions (PUTONs) which transforms the initial situation into a situation in which block A is ON block B. Such a sequence may be derived from a constructive proof of the statement (EXISTS (S) (TRUE (ON A B) S)) from the initial situation.[Construction]

One strategy is to derive all possible consequences of the axioms using the logical rules of inference without SHOW restrictions. If the goal state is a possible future of the initial state, then a solution sequence will eventually be generated. This forward chaining strategy generates piles of irrelevant states which, although accessible from the initial state, are not on any solution path to the goal state.

A dual strategy is backward chaining. This can be accomplished using the SHOW rules described previously to generate all possible pasts of the goal state. Although all the states so generated are relevant to the goal, most of these are inaccessible from the initial situation.[Subgoal Filters]

Refinement planning is the strategy of decomposing the problem into the sequential attainment of intermediate "islands" or subproblems.[Islands] Both forward and backward chaining are special cases of this strategy, in which the islands proposed are derived by finding states separated from the initial or goal states by the application of a single operator. The more general use is to propose subproblems which are not necessarily immediately accessible from the initial or goal states, but which, if solved, enormously restrict the size of the remaining subproblems. These intermediate subgoals are produced at the risk of being either irrelevant to the goal or impossible to achieve from the initial state, and so must be suggested by "methods" which "know" reasonable decompositions of a domain-specific nature.[HigherSpaceBC]

Several additional constraints influence the selection of problem solver strategies. Many operator sequences have no net effect (they are composite "no-ops"). A problem solver which fails to recognize that these sequences produce no change of state will loop unless its search is globally breadth-first. In addition, it will waste effort deriving solutions to problems isomorphic to ones it has already solved. To solve this problem, it is important to represent the properties of situations in such a way that two situations which are identical with respect to some purpose can be recognized as such.

Implementation of a Refinement-Planning Problem Solver

The principal difficulty of solving problems in worlds which can have arbitrarily many states is that any simple deduction mechanism will explore all of them. Our problem solver limits the potential combinatorial explosion by having domain specific rules which control the introduction of new states. The problem solver also contains rules which are domain independent, of which Modus Ponens and Conjunction Introduction are examples. These SHOW rules will only be invoked for questions which concern an existing state. They are not allowed to generate new or hypothetical states.

The statement (GOAL <condition> <situation>) is asserted when we want <condition> to be achieved in some situation which is a successor of <situation>. The following rule is triggered by this assertion and controls the solution process. When the goal is satisfied this rule will assert (SATISFIED (GOAL <condition> <situation>) <new-situation>) where <new-situation> is the name of the situation where <condition> now holds. The subrule first checks whether the goal is already true. It asserts (TRUE? <condition> <situation>) (asking the question, "Is <condition> true or false in <situation>?") and sets up two rules which wait for the answer. A convention of our rules is that an answer is guaranteed. Processing the goal will continue when the

answer (YES or NO) is asserted. If the condition is not true in the current situation, planning proceeds by asserting (ACHIEVE <condition> <situation> <goal>), where <goal> is the fact name of the goal. The ACHIEVE facts trigger the relevant methods for achievement of the goal condition. These methods may introduce new situations and perform actions. If a method thinks that it has succeeded in producing a successor state of the given situation in which the goal condition is true it asserts (ACHIEVED? <goal> <new-situation>), where <new-situation> is the situation in which <goal> is thought to be satisfied. The goal rule then checks this suggestion with TRUE? and makes the SATISFIED assertion if successful. If the method is in error, the bug manifestation is noted in a BUG assertion. This is marked as a contradiction and causes backtracking. A more sophisticated problem solver would at this point enter a debugging strategy. The justification of the contradiction can be traced. This information is helpful in diagnosing the fault and constructing a patch to the domain specific methods.

```
(Rule (:g (GOAL :c :s))
      (Assert (TRUE? :c :s) (Goal-true? :g))
      (Rule (:q (TRUE? :c :s))
            (Rule (:t (YES :q))
                  (Assert (SATISFIED (GOAL :c :s) :s)
                          (Goal-immed-satisfied :g :t)))
            (Rule (:t (NO :q))
                  (Assert (ACHIEVE :c :s :g) (Goal-unsatisfied :g :t))
                  (Rule (:w (ACHIEVED? :g :s1))
                        (Assert (TRUE? :c :s1) (Did-it-succeed? :g :t :w))
                        (Rule (:q2 (TRUE? :c :s1))
                              (Rule (:f (YES :q2))
                                    (Assert (SATISFIED (GOAL :c :s) :s1)
                                            (Win :g :f)))
                              (Rule (:f (NO :q2))
                                    (Assert (BUG :g :w :f)
                                            (Contradiction :w :f)))))))))
```

To check whether a statement is true in a situation the SHOW mechanism is used. The assertion of (GOAL <condition> <situation>) requests <condition> to be true in some successor state of <situation>. The statement <condition> must be relative to a situational variable because it is checked in two (potentially) different situations in the GOAL rule above. In order to test whether this statement is true or false this variable must be bound to the particular situation being considered. The condition of a GOAL assertion must be of the form (L <variable> <predicate>), ("L" abbreviates "LAMBDA") where <predicate> is a predicate form with an open situational variable <variable>. The unification of the trigger pattern of TRUE? with the assertion (TRUE? <condition> <situation>) has the effect of binding the particular situation <situation> being considered with the situational variable <variable> used in <predicate>. By lambda-

abstracting the goal condition, we eliminate the explicit mention of any particular situation in the goal description. "Equivalent" goals are variants, and so will be identified by the AMORD interpreter.[Equivalent Goals]

```
(Rule (:g (TRUE? (L :s :p) :s))
      (Rule (:f :p)
            (Assert (YES :g) (Return :g :f)))
      (Rule (:f (NOT :p))
            (Assert (NO :g) (Return :g :f)))
      (Assert (SHOW :p) (Try-positive :g))
      (Assert (SHOW (NOT :p)) (Try-negative :g)))
```

If the goal is a conjunction of conditions, the following rule is triggered. Some conjunctive goals can be achieved by achieving each conjunct separately. This is called a LINEAR-PLAN. Sometimes the conjuncts can be achieved in one order but not in the other order.[PCBG] A conjunctive goal cannot always be decomposed in this way.[Anomalous] In the case of such a non-linear problem, our rule fails.

```
(Rule (:f (ACHIEVE (L :s (AND :c1 :c2)) :s1 :purpose))
      (Assume (LINEAR-PLAN :f) (First-order :f))
      (Rule (:p (LINEAR-PLAN :f))
            (Assume (STATED-ORDER :p) (Conjunct-order :p))
            (Rule (:o (STATED-ORDER :p))
                  (Assert (ORDERED-PLAN :s :c1 :c2 :s1 :purpose) (Try :o)))
            (Rule (:o (NOT (STATED-ORDER :p)))
                  (Assert (ORDERED-PLAN :s :c2 :c1 :s1 :purpose) (Try :o))))
      (Rule (:p (NOT (LINEAR-PLAN :f)))
            ;This problem solver has no clever ideas about this case.
            (Assert (FAIL :p) (Contradiction :p))))
```

The next rule refines a conjunctive goal as an ordered linear plan. It produces the subgoal of finding an "island" in which the first conjunct is true. If the first subgoal can be satisfied, it then establishes the subgoal of satisfying the second conjunct in a successor of this island. If the second subgoal can be satisfied the resulting state is proposed as a solution to the conjunctive goal. The goal rule which triggered this method is resumed by the statement (ACHIEVED? <purpose> <new-situation>) which tests (using TRUE?) the original goal condition in <new-situation>. If this method is wrong, the GOAL rule will fail.

```
(Rule (:f (ORDERED-PLAN :s :c1 :c2 :s1 :purpose))
      (Assert (GOAL (L :s :c1) :s1) (Subgoal-1 :f))
      (Rule (:sat1 (SATISFIED (GOAL (L :s :c1) :s1) :s2))
            (Assert (GOAL (L :s :c2) :s2) (Subgoal-2 :f :sat1))
            (Rule (:sat2 (SATISFIED (GOAL (L :s :c2) :s2) :s3))
                  (Assert (ACHIEVED? :purpose :s3) (Win? :f :sat1 :sat2)))))
```

Methods redundantly incorporate knowledge included in the axioms. They embody the domain specific heuristics for constructing effective subgoals. The following rule suggests that to achieve (ON A B) one should first achieve a situation which A has a cleartop and B has space for A. From this situation, we can immediately (PUTON A B) producing a situation in which the goal is achieved. This method is the only rule which creates new situations.

```
(Rule (:f (ACHIEVE (L :s (TRUE (ON :a :b) :s)) :s1 :purpose))
      (Assert (GOAL (L :x (AND (TRUE (CLEARTOP :a) :x)
                               (TRUE (SPACE-FOR :a :b) :x)))
                    :s1)
              (Prerequisite-for-PUTON :f))
      (Rule (:sat (SATISFIED
                   (GOAL (L :x (AND (TRUE (CLEARTOP :a) :x)
                                    (TRUE (SPACE-FOR :a :b) :x)))
                         :s1)
                   :s2))
            (Assert (ACHIEVED? :purpose ((PUTON :a :b) . :s2))
                    (Record-PUTON-purpose :f :sat))))
```

The following rules describe methods for achieving each predicate of the domain and its negation. To achieve NOT-ON, move the offending object to the table.

```
(Rule (:f (ACHIEVE (L :x (NOT (TRUE (ON :a :b) :x))) :s1 :purpose))
      (Assert (ACHIEVE (L :u (TRUE (ON :a Table) :u)) :s1 :purpose)
              (Get-rid-of :f)))
```

To make space on something, achieve NOT-ON for all offending objects.

```
(Rule (:f (ACHIEVE (L :s (TRUE (SPACE-FOR :a :y) :s)) :s1 :purpose))
      (Rule (:o (TRUE (ON :x :y) :s1))
            (Assert (ACHIEVE (L :u (NOT (TRUE (ON :x :y) :u))) :s1 :purpose)
                    (Make-space-for :f :o))))
```

To clear a block, achieve NOT-ON for all other blocks on it.

```
(Rule (:f (ACHIEVE (L :s (TRUE (CLEARTOP :y) :s)) :s1 :purpose))
      (Rule (:o (TRUE (ON :x :y) :s1))
            (Assert (ACHIEVE (L :u (NOT (TRUE (ON :x :y) :u))) :s1 :purpose)
                    (Make-CLEARTOP :f :o))))
```

The methods introduce some incompleteness that was not present in the original axioms. In return the problem solver always halts by running out of further rules to run. The main reason the specific methods could be used successfully is that the deductions are explicitly controlled by control assertions (GOAL, ACHIEVE, ACHIEVED?, TRUE?).

Conclusions

Many kinds of combinatorial explosions can be avoided by a problem solver that thinks about what it is trying to do. In order to be able to meditate on its goals, actions and reasons for belief, these must be explicitly represented in a form manipulable by the deductive process. In fact, this "internal" control domain is a problem domain formalized using assertions and rules just like an "external" domain. How can we use assertions about control states to effectively control the deductive process?

The key to this problem is a set of conventions by which the explicit control assertions are used to restrict the application of sound but otherwise explosive rules. These conventions are supported by a vocabulary of control concepts and a set of systemic features. The applicability of a rule can be restricted by embedding it in a rule having a pattern which matches a control assertion as an entrance condition. The rule language allows the variables bound by matching the control assertions to further restrict the embedded rule. But we want the conclusions of sound rules to depend only on their correct antecedents and not on the control assertions used to restrict their derivation. This is necessary to enable fruitful deliberations about the reasons for belief in an assertion. The system must provide means for describing the reasons for belief in an assertion and means for referring to an object of belief.

Sometimes it is necessary to make assumptions -- to accept beliefs that may later be discovered false. Conclusions of rules which operate with incomplete knowledge must depend upon the control assumptions made. Accurate dependencies allow precise assignment of responsibility for incorrect beliefs. This is necessary for efficient search and perturbation analysis.

Notes

Advice Taker

The term "Advice Taker" originates with McCarthy [1968].

Additive Knowledge

Is it possible to have an additive system in which knowledge about other knowledge can be expressed? Sometimes advice may be negative. For example, the process of sorting a list may be defined as finding a permutation of the list which is ordered. An obvious procedure derived from this definition is that of enumerating the permutations of the list and testing each for order. If better methods become known, we will want to give the advice that this method stinks. How can this be an additive piece of knowledge? Perhaps a way to make such knowledge additive is to formalize the "state of mind" of the problem solver, and let such advice change its state of mind.

Procedures

The "Procedural Embedding of Knowledge" is the philosophy popularized by Winograd [1972] and Hewitt [1972, 1975] that knowledge can be most profitably represented as computer programs.

Coherence

In *Human Problem Solving* [Newell and Simon 1972], the apparently coherent behavior of human subjects is also explained in terms of a set of relatively independent agents (formalized as productions). Minsky [1977] proposes a structure for human behavior in terms of a "society of agents".

EL

EL is a set of rules for electrical circuit analysis which embodies the method of propagation of constraints. [Sussman and Stallman 1975] EL is implemented in ARS. [Stallman and Sussman 1976]

Simon's Ant

Simon [1969] points out that apparently complex behavior can result from simple procedures operating in a complex but constraining domain.

Explicit Control

Production System devotees have tended to approach the problem of control through the architecture of the machine supporting the problem solver. Most of these studies are concerned with devices like production ordering, recency criteria for working memory elements, and priority measures on productions and memory elements. [Hayes-Roth and Lesser 1977, McDermott and Forgy 1976]

Drew McDermott's [1976] NASL interpreter explicitly records control assertions in guiding an electronic circuit design program. His system also records some dependency

information, but does so in an automatic fashion rather than explicitly. Other uses of explicit representations of the problem solver control state are used in NOAH [Sacerdoti 1975] and MYCIN [Davis 1976]. Production System problem solvers (such as GPSR [Rychner 1976]) necessarily record goals explicitly in working memory, but only as an implementation of standard consequent reasoning, rather than as a mechanism for careful control.

Well-Founded Support

Means for effectively maintaining a well-founded justification for each believed fact have been investigated by Doyle [1977]. His system TMS (Truth Maintenance System) has been incorporated in our design of AMORD.

Backtracking

Dependency-Directed Backtracking is a technique for careful backtracking which was introduced by Stallman and Sussman [1976] in the context of electrical circuit analysis.

Separation

Hayes [1973], Kowalski [1974], and Pratt [1977] have advocated the separation of problem solving knowledge into "competence" and "performance" components. We feel that this is the wrong distinction to make, as the competence knowledge must necessarily be replicated in the performance knowledge. Our proposed methodology requires these forms of knowledge to be integrated for efficient control, but separated by explicitly recorded dependencies.

Micro-PLANNER

Micro-PLANNER [Sussman, Winograd, and Charniak 1970] was a language based on Hewitt's [1972] PLANNER which had a pervasive system of chronological backtracking.

Control Vocabulary

McDermott [1977] argues that it is the vocabulary used in explicit (production-like) control that is important, not the specific machine architecture. See also the detailed vocabularies he [McDermott 1976] and Sacerdoti [1975] develop for talking about tasks and actions.

Suppression

Of course, redundant conjuncts can be suppressed by building the semantics of conjunction into the problem solver. This just puts off the problem as non-primitive relations can also explode in this fashion. Resolution [Robinson 1965] and associated combinatorial strategies are domain independent techniques for suppressing some combinatorial explosions while maintaining completeness.

Implications

It is sometimes necessary to derive an implication as a subgoal, as in a conditional proof in natural deduction. In this example we have not provided for such subgoals. The details of conditional proof in the context of a truth maintenance system are described by Doyle [1977].

AMORD

A Miracle of Rare Device, a name taken from S. T. Coleridge's Kubla Khan. AMORD has been implemented in SCHEME [Sussman and Steele 1975], a lexically-scoped dialect of LISP with tail recursion.

Variants

In AMORD, facts are indexed so that variant statements of a fact are identified. If a fact is derived in different ways, it is justified by each of the various derivations. These multiple justifications are useful if a derivation is later withdrawn. [Stallman and Sussman 1976, Doyle 1977]

Unifies

Our matcher is based on the unification algorithm used in resolution, but ignores the restrictions of first order logic. In particular, there are no distinguished symbols or positions.

Order

The AMORD language does not specify the order in which rules are executed. The order is irrelevant to the discussion in the text.

Nogood

A summary of the reasons for a contradiction which are independent of a particular set of assumptions is used in ARS [Stallman and Sussman 1976] to restrict future choices. Doyle [1977] extended and clarified this notion and its relationship to the logical notion of conditional proof.

Situations

The situational calculus formalizations of changing world was introduced by McCarthy [1968], and further developed by McCarthy and Hayes [1969]. This technique is closely related to the methods of modal logic. [Kripke 1963]

Construction

Green's [1969] method of constructive proof did not take into account the initial situation. For example, one should state the goal: (EXISTS (S) (AND (FUTURE INIT S) (TRUE (ON A B) S))) where FUTURE is defined according to the operators that are applicable.

Subgoal Filters

Some impossible subgoals can be pruned by using an external semantic filter. Gelernter's [1963] Geometry Machine uses an analytic geometry diagram for this purpose.

Islands

The concept of refinement planning with islands was introduced by Minsky [1963].

HigherSpaceBC

Refinement planning might be viewed from the GPS [Ernst and Newell 1969] and ABSTRIPS [Sacerdoti 1974] perspective as backward chaining in a higher level space which controls the activities in the action sequence space.

Equivalent Goals

We mean equivalences relative to situational variables. There are other types of equivalences which are not caught by this technique, such as that between the goals (AND A B) and (AND B A).

PCBG

One way in which a solution of a conjunction by a linear plan may be incorrect is the Prerequisite-Clobbers-Brother-Goal bug discussed by Sussman [1974, 1975]. This bug may be fixed by reordering the plan. Other related bugs in the world of fixed-instruction turtle programs are discussed by Goldstein [1974].

Anomalous

Allen Brown [Sussman 1975] discovered that it is impossible to use a linear plan to construct (AND (ON A B) (ON B C)) if the initial situation contains (ON C A). Tate [1974], Sacerdoti [1975], and Warren [1974] have proposed several solutions to this problem.

Bibliography

[Davis 1976]
Randall Davis, "Applications of Meta Level Knowledge to the Construction, Maintainance and Use of Large Knowledge Bases," Stanford AI Lab Memo AIM-283, July 1976.

[Doyle 1977]
Jon Doyle, "Truth Maintenance Systems for Problem Solving," MIT AI Lab TR-419, 1977.

[Ernst and Newell 1969]
George W. Ernst and Allen Newell, *GPS: A Case Study in Generality and Problem-Solving*, Academic Press, New York, 1969.

[Gelernter 1963]
H. Gelernter, "Realization of a Geometry-Theorem Proving Machine," in Feigenbaum and Feldman, *Computers and Thought*, pp. 134-152.

[Goldstein 1974]
Ira P. Goldstein, "Understanding Simple Picture Programs," MIT AI Lab, TR-294, September 1974.

[Green 1969]
C. Cordell Green, "Theorem-Proving by Resolution as a Basis for Question-Answering Systems," in Meltzer and Michie, *Machine Intelligence 4*, pp. 183-205.

[Hayes 1973]
P. J. Hayes, "Computation and Deduction," Proc. MFCS, 1973.

[Hayes-Roth and Lesser 1977]
F. Hayes-Roth and V.R. Lesser, "Focus of Attention in the Hearsay-II Speech Understanding System," CMU CS report, January 1977.

[Hewitt 1972]
Carl E. Hewitt, "Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot," MIT AI Lab, TR-258, April 1972.

[Hewitt 1975]
Carl Hewitt, "How to Use What You Know," *IJCAI 4*, September 1975, pp. 189-198.

[Kowalski 1974]
Robert Kowalski, "Logic for Problem Solving," University of Edinburgh, Department of Computational Logic, DCL Memo 75, 1974.

[Kripke 1963]
S. Kripke, "Semantical Considerations on Modal Logic," *Acta Philosophica Fennica*, 83-94, 1963.

[McCarthy 1968]
John McCarthy, "Programs With Common Sense," in Minsky, *Semantic Information Processing*, pp. 403-418.

[McCarthy and Hayes 1969]
J. McCarthy and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in Meltzer and Michie, *Machine Intelligence 4*, pp. 463-502.

[McDermott 1976]
Drew Vincent McDermott, "Flexibility and Efficiency in a Computer Program for Designing Circuits," MIT AI Lab TR-402, December 1976.

[McDermott 1977]
Drew McDermott, "Vocabularies for Problem Solver State Descriptions," *IJCAI 5*, August 1977.

[McDermott and Forgy 1976]
J. McDermott and C. Forgy, "Production System Conflict Resolution Strategies," CMU CS report, December 1976.

[Minsky 1963]
Marvin L. Minsky, "Steps Toward Artificial Intelligence," in Feigenbaum and Feldman, *Computers and Thought*, pp. 406-450.

[Minsky 1977]
Marvin L. Minsky, "Plain Talk about Neurodevelopmental Epistemology," *IJCAI 5*, August 1977.

[Newell and Simon 1972]
Allen Newell and Herbert Simon, *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Pratt 1977]
Vaughan R. Pratt, "The Competence/Performance Dichotomy in Programming," MIT AI Lab Memo 400, January 1977.

[Robinson 1965]
J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *JACM, 12* (January 1965), pp. 23-41.

[Rychner 1976]
Michael D. Rychner, "Production Systems as a Programming Language for Artificial Intelligence Applications," CMU CS Report, December 1976.

[Sacerdoti 1974]
Earl D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, Vol. 5, No. 2, pp. 115-135.

[Sacerdoti 1975]
Earl D. Sacerdoti, "A Structure for Plans and Behavior," SRI AI Center, TN 109, August 1975.

[Simon 1969]
Herbert Simon, *The Sciences of the Artificial*, MIT Press, Cambridge, Massachusetts, 1969.

[Stallman and Sussman 1976]
Richard M. Stallman and Gerald Jay Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," MIT AI Memo 380, September 1976.

[Sussman 1974]
Gerald Jay Sussman, "The Virtuous Nature of Bugs," *Proc. AISB Summer Conference*, July 1974.

[Sussman 1975]
Gerald Jay Sussman, *A Computer Model of Skill Acquisition*, American Elsevier Publishing Company, New York, 1975.

[Sussman and Stallman 1975]
Gerald Jay Sussman and Richard Matthew Stallman, "Heuristic Techniques in Computer-Aided Circuit Analysis," *IEEE Transactions on Circuits and Systems*, Vol. CAS-22, No. 11, November 1975, pp. 857-865.

[Sussman and Steele 1975]
Gerald Jay Sussman and Guy Lewis Steele Jr., "SCHEME: An Interpreter for Extended Lambda Calculus," MIT AI Lab Memo 349, December 1975.

[Sussman, Winograd and Charniak 1970]
Gerald Jay Sussman, Terry Winograd and Eugene Charniak, "MICRO-PLANNER Reference Manual," MIT AI Lab, AI Memo 203.

[Tate 1974]
Austin Tate, "INTERPLAN: A plan generation system which can deal with interactions between goals," University of Edinburgh, Machine Intelligence Research Unit, MIP-R-109, December 1974.

[Warren 1974]
David H. D. Warren, "WARPLAN: A System for Generating Plans," University of Edinburgh, Department of Computational Logic Memo No. 76, June 1974.

[Winograd 1972]
Terry Winograd, *Understanding Natural Language*, Academic Press, New York, 1972.